20 | 高可用设计，让产品没有后顾之忧

2020-12-09 乔新亮

乔新亮的CTO成长复盘 进入课程 >

讲述：乔新亮

时长 22:38 大小 20.73M

你好，我是乔新亮。这一讲，我想和你聊聊，关于高可用设计的那些事儿。

一提起高可用设计，很多同学立刻就会想到“冗余设计”、“故障转移”等关键词。确实，在大部分与高可用相关的分享里，这两个词往往会被重点强调。

所谓“冗余设计”，是指要通过集群来替代单点服务，做好冗余备份。单点架构是高可用的大敌，“把鸡蛋放在不同的篮子里”是高可用最朴实、最有效的设计思路之一；“故障转移”则是为了缩短故障时间，保证故障发生时，业务可以快速恢复。

如果你对高可用设计了解得更多，可能还会给出如“CAP 理论”、“异地多活”、“双机架构”一样的更多概念，并且详细解释应用方法。

像这样面对问题，能立刻在大脑知识地图中检索并给出应对举措或方案，是一个优秀的特质，能帮助你通过许多高难度面试。但不知你有没有想过，**为何自己在面试中对答如流，在实际工作中却仍然会遇见许多高可用难题呢？为什么还是无法主导一家大型企业整体架构的高可用设计呢？**

问题或许是多方面的，比如企业现阶段的发展状况特殊，缺乏实战机会，导致个人资历不够；或者相关部门不太配合，组织人员能力差，代码 Bug 太多，让你一想就气不打一处来……

以上原因都有可能，也都客观存在于很多公司。但我认为，最重要的原因是，缺乏对高可用设计自顶向下的、比较通透的理解和认知，因而在实践中常常迷失在技术细节里，难以窥见架构的全貌。

那么这一讲，我们就来尝试一下，看能否通过一些简单的梳理，建立对高可用设计的完整认知。

解剖高可用设计

在架构设计部分，我们讲了：粗略地说，想做好架构设计，第一步是将一个 IT 系统从应用层级至底层基础设施，全部拆解为一个个应用模块，我们也可以称之为“元素”或“组件”；第二步是保证各个模块间不能孤立存在，还要做好充分的协作，协作通过应用模块对外暴露的“服务”来承载，我们也可以称之为“连接”。

所以说，应用模块和服务，或者叫做元素和连接，共同组成了所谓的架构。那么，要实现架构的高可用，就意味着实现所有元素、连接的高可用。

至此，其实我们已经得出了一点非常重要的认知，再重复一遍：**真正的高可用，是指实现所有元素、所有连接的高可用。只要一个元素或一个连接没有做高可用设计，都意味着风险的存在。**

比如，公司要求每天必须按时上班，只要迟到就罚钱，那么你如何保证自己的“出勤系统”是高可用的呢？一定是全链条做高可用设计。闹钟至少有两个——互为主备、要保证衣物都在、要为堵车准备预案、要保证高峰期能挤上电梯……你甚至得保证身体健康，不然可能因为拉肚子而迟到。

要注意，在这个例子里。闹钟可能不响、堵车可能会发生，没问题，只要在老板眼里，你永远按时打卡就行，只要整个系统是高可用的就好。**更准确地说，所谓高可用，是要保证“业务的连续性”，即在用户眼里，业务永远是正常（或基本正常）对外提供服务的。**

怎么样，是不是有点难？是的，在实际工作中，大部分企业的架构设计没有做到高可用，原因可能只有两个：

1. 相关负责人压根儿就没考虑高可用设计；
2. 做全套高可用的代价太大了，钱不够用，时间不够用。

先前有位同学在专栏下方留言说，乔老师传达的理念就是“**trade-off**”。我一看，禁不住为他鼓掌喝彩——说得太对了。在企业层面，很多难题不是“简答题”，而是“选择题”。所以在专业成长这一章，我们会先讲“架构决策”，再讲其他内容。

其实人生也如此，职业生涯也如此，如果都有唯一的正确答案，那太简单了。很多教程回答的都是“唯一正确答案”，听着很有道理，实际做的时候，发现根本不是这么一回事。

作为成年人，我们必须得清醒地认识到，人生这道选择题，根本没有标准答案，你能做的只有在不完美中寻找完美，不断 trade-off。

做好决策，在很多情况下，并不意味着风险就消失了 —— 风险始终存在。根据墨菲定律，如果事情有变坏的可能，无论概率有多小，它一定会发生。那么假设你我就是企业架构的总负责人，现在要保证企业整体业务的连续性，应该怎么办呢？

有一个很通用的办法，叫做“**冗余设计**”（我们在文章开头提过了）。所谓“**集群**”、**“分布式”**，这些名词大体上都是在描绘“冗余设计”的概念。

但冗余设计也不是万能的，它只能解决底层物理机器，也就是某一个实例的不可用问题。如果是代码逻辑出现了问题，冗余设计就失效了 —— 一个恶性 Bug 在生产环境爆发，后果是所有集群都会遭殃。

没钱做 100% 的高可用设计，又想尽量提供系统的抗风险能力。作为架构负责人，应该怎么办呢？

这里你可以停下一分钟，略微思考一下。

第一个解决方案是，在风险爆发、系统出现问题的情况下，对外提供“降级服务”。也就是说，当团队没有时间去设计、测试并确保当前组件高可用时，如果因为未知原因，导致组件出现故障，我们需要提供一个逻辑相对简单，并且一定可用的服务替代原来的服务实现，实现服务对外的高可用。

以电商系统的物流时效产品为例，正常情况下，该系统需要通知用户已购商品的到达日期。比如，邻近区域当日送达、跨省隔日送达、偏远地区三日送达，等等。具体的物流时效和用户和仓库的距离、仓库作业能力、车辆配送能力很多因素都有关。

但如果因为一个 Bug 或一个配置文件修改错误，导致物流时效产品崩溃了，用户在四级页、购物车，订单确认页调用该服务时失败，怎么办呢？

一个解决办法是：由服务器端技术人员写一段程序，保证任何用户查询物流情况，都显示 3 天送达。出问题的时候，使用此服务进行替换。然后，全体技术人员努力恢复服务，系统恢复后替换回正常的物流时效服务（如果有兴趣，这里你可以想想，为什么这段代码应该是服务端技术人员来写？答案藏在架构设计一节）。

看到这里，你可能就笑了：老乔你这是欺骗用户呢，这还叫什么高可用？

可这就是“降级服务”，实打实的高可用保障手段。在用户眼里，业务是连续的，只是可靠性降低了。其实对于架构师而言，高可用和高可靠应该是两个不同的概念，只是很多人将其混为一谈。

在最理想的情况下，我们既保证高可用，也保证高可靠；但出现问题时，我们优先保证高可用，其次保证高可靠。这是另一点关键认知。

其实在不同领域里，有很多类似的做法。比如在流媒体领域，当用户观看直播出现严重卡顿时，很多企业的第一反应不是查服务器 Log，而是为用户自动降码率。因为比起画质降低，卡的看不了显然会让用户更痛苦。

如果“降级服务”还解决不了问题，应该怎么办？答案是提供“熔断服务”，让出现 Bug 的模块从系统中“熔断”，虽然用户会看到物流系统报错，但整个业务依然是正常响应

的，不会被一个系统的 Bug 拖死。

现在我们总结一下：**高可用意味着对系统全部元素、连接都进行高可用设计，在物理实例层面主要表现为冗余和集群设计，在代码逻辑层面，方法则多种多样。当你的资源和精力不足以实现全链路高可用时，提供“降级服务”和“熔断服务”，优先保证高可用，其次保证高可靠。**

企业里面有些核心服务是不能降级的，对于这类服务，就一定要通过研发流程管理，确保服务的高可用、高可靠。一名合格的技术管理者，要能够识别核心服务，并引导团队重点关注。

高可用，不只是个“设计问题”

在我的职业生涯中，有两次生产事故让我印象最为深刻。

一次是机房停电，一次是知名开源软件 ZooKeeper 出现严重 Bug（几年前的一个版本 Bug，连接数超过阈值就会停止服务，不知道现在是不是修复了）。这两次事故发生时，我基本都处于焦头烂额的状态，甚至几天几夜不能睡觉。它们一个属于底层物理实例出现了问题，一个属于代码逻辑出现了问题。

对于你而言，我觉得代码逻辑导致的系统故障可能更为常见，机房停电、爆炸、地震毕竟发生几率比较小。

如果将“放大镜”对准代码逻辑导致的系统“不可用”问题，我们就会发现**高可用设计真正的敌人是“变化”。**

设想一下，如果生产环境不发生变化 —— 不发布新版本、不修改配置文件、不修改数据库脚本，系统大概率会一直保持正常（你可能会说，老乔，服务器压力激增也会导致问题。前面我们讲了，这属于架构设计中的“流控”设计，通过容量规划设计，流量控制问题很容易解决）。

此时，生产环境发布新版本就会成为一个影响巨大的变量，极有可能对系统的可用性造成挑战。

所以，**研发管理水平的高低，决定了你在版本发布方面的成功率和信心。**

单就版本发布问题来说，你需要关注研发管理的三个关键点：

1. 记录系统的任何一次发布和变化，包括发布系统 / 组件、发布时间等；确保自己可以随时定位任何一个时间段内的任何元素及任何发布动作，包括但不限于代码、配置文件、SQL 脚本、设备参数修改等；
2. 发布时不影响业务；
3. 保证任何发布都可以回滚。尤其当一个大版本的发布时，能否精确识别回滚单元，并做到秒级回滚。

只要做到以上三点，哪怕你在上午十点发布新版本，又有什么关系？根本就不必等到半夜嘛。

当然，总是回退也不是办法。作为一个研发组织，我们还可以从另外一个角度，提高系统的抗风险能力。这里，我要先问你两个问题，如果有时间的话，请你思考一下：

1. 由代码逻辑导致的系统风险，是如何进入生产环境的？
2. 生产环境出现严重故障，是不是毫无征兆地发生的？

首先回答第一个问题，风险是经由开发环境、SIT 环境、压测环境、PRE 环境，进入生产环境的。所以我们要做的，是严格检查各个环境下的异常，**研发管理规范，应该为代码版本进入下一个环境设置准入标准，对于任何异常，都有负责人进行修正。如果异常通过了评估，审核者要对其负责**。

第二个问题，答案当然是否定的。就像人在生病前，会在各项生理指标上有所表现。**系统 Bug 在导致生产故障前，也往往会有各类异常，我们要做好监控并正式的处理掉它**。

这里我想说点题外话，在我们这个技术行业，每年都有猝死的情况发生，每次看到这种消息，我都觉得很遗憾。大家要关注自己的身体健康，关注身体给出的各种警告，爱自己、照顾好自己，才能爱家人。只要关注自己的健康、想办法，猝死大概率不会发生的。

回归正题，在极少数情况下，一个严重 Bug 会藏在生产环境里，始终没有触发。但当它被触发时，可能就会导致生产环境“暴毙”。前面我讲的关于 ZooKeeper 的例子就是这样，到连接数超过阈值后，系统突然就挂了，所有人措手不及。

对于这种情况，一旦碰到了，某种程度上就要认命。我一直讲，人生，运气也很重要，非常重要。如果已经尽全力了，还是出现了问题，要学会接受自己，承认自己的运气不好，将这当作是成功路上的经验、教训，这才是真正的成长性思维。拒绝不完美，渴望成长，当然很棒，但很多心理问题的源头也在这里，要学会接受自己。

所以，开源软件其实是存在一定风险的。但开源软件依然代表了软件研发行业的一种主要潮流，毕竟你免费用了人家的软件那么久，有些许风险不是很正常吗？所以我依然允许团队向生产系统引入开源代码，只是从那次事故以后，我要求：引入开源代码的技术人员，必须通读并掌握其代码。

还有许多研发管理方面的注意事项，包括代码 Review 文化、DevOps 等，我们就不再展开细说了。**关键在于要注意，真正的、为业务负责的高可用设计，不是画框图就能解决的，它是一个面向 IT 组织的整体设计。**

结语

到这里，我们这一讲的内容就接近尾声了。

高可用设计，意味着“Design For Failure”，最重要的是让我们做产品没有后顾之忧。如果后院天天起火，研发团队每天胆战心惊，也就无从谈起“将产品打磨至卓越”了。所以，做好高可用，一定程度上就是在实践“慢就是快”的认知理念。

如果你有仔细揣摩以上内容，或许会发现：虽然我们或许不能保证所有服务高可靠，但我们是可以保证所有服务高可用的。其关键点在于，面向所有的元素和连接，都要做设计。任何没有被设计过的元素和连接，往往都是不可靠的。

任何一个要达到的目标，都要被量化；任何一个量化的目标，都是一个契约，要有契约精神；任何一个契约，都要进行设计。没有设计的内容，怎么可能如你所愿呢？何况，什么才是你的“所愿”呢？

从这个角度来看，做一名卓越的架构师，真是个苦差事，但也充满了乐趣。

你可能也会想，老乔，那我学的 CAP 理论、异地多活还有用吗？是不是都白学了？

这些当然是有用的，但它们都有具体的使用场景限制。比如对于 CAP 理论来说，在分布式基础设施层的应用有些用，在应用层就受限了，许多人是学完了就忘了，设计架构时该怎么干还怎么干；至于异地多活，已经超过了国内大部分企业的现阶段的业务需要。而且我认为，异地多活最大的优势在于，对架构可扩展性和可维护性的提升，而不仅仅是高可用。

学习，尤其是学习技术和管理知识，其实要有“功利心”。学完了一定要去实践，这样才能真正化为己用。我总是跟你讲：持续学习、学以致用、终身成长，有太多同学就是没做到学以致用，才导致“学了也白学”的结果出现。

希望你在以后的学习生活中，能始终做到学以致用，保持高速的成长，终身成长。

我们下一讲再见！

10 人觉得很赞 | 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 19 | 产品思维，契约精神是基础，洞察人性才能成就卓越

下一篇 21 | 高性能设计，一切都围绕着契约精神

精选留言 (7)

写留言

huaxiaofeng
2020-12-09

人体就是一个高可用的系统，人的很多器官有两个及以上(冗余设计)，少了一个，人体还能够正常的工作，还能继续生活，每个器官也有不同的等级，就好比服务治理中的，核心服务，和非核心服务，核心服务不能挂，挂了就会影响到整个业务，就好比人的心脏，如果停止工作，那整个人就挂了，但是断了一只手，人还是能够生活(服务降级)，架构设计可以参考生物学中的很多概念。

展开

作者回复: 你好， huaxiaofeng
这个比喻太好了，真棒

5

glutton
2020-12-09

每天上班的例子，真是太好了，贴切又联系生活，很好理解。

高可用不等于高可靠。
高可用的本质是冗余设计，实例级、机房级、城市级等等；代码逻辑的bug，通过灰度、沙箱、预发布等多环境验证；如果是稳定元素突发事件，那就认命，接纳自己，这个认...

展开

作者回复: 你好，glutton
可以啊，大家认知一致，做事简单，欢迎加入，你可以留下联系方式，我来联系你。

1 2

46
2020-12-09

讲的非常好，超值的课程！

展开

作者回复: 你好，😁 46
谢谢

1

李潘
2020-12-09

非常同意下面的观点：
1. 高可用就是保证业务连续性。
2. 高可用不等于高可靠，高可用会有很多trade-off。
3. 高可用的实现，需要架构自顶向下的进行思考，只单独考虑某个功能或者模块的高可用，并不能保证整个业务链路的高可用。...
展开

作者回复: 你好，李潘
总结的特别好

1

Demon.Lee
2020-12-11

重点：学以致用，学以致用，否则学了也是白学。从小学到大学，没用过的是不是都还给老师了？太真实。

作者回复: 你好，Demon.Lee
哈哈，加油

剑八
2020-12-10

高可用就是出现异常时业务能不中断
可能是备案

作者回复: 你好，剑八
业务连续性，系统可以出问题，业务不能出问题，业务要连续，也就是你说的不中断。

spark
2020-12-09

高可用是系统弹性能力和余量问题。系统有弹性和余量才能抗风险。人和系统一样，有时候比的是谁有弹性和余量，单讲具体能力大家都差不多。

展开

作者回复: 你好，spark
在局部能力差不多的情况下，就要拼资源，是这样的。所以在商场竞争中有时候资本的力量很厉害也是同样的道理。